Using IOT for I/O and MPI Performance Analysis

NAS Webinar

Sept 6, 2017

NASA Advanced Supercomputing Division

MPI and I/O Analysis Tools at NAS

- Mpiprof
  - In-house profiling tool
  - Extremely light-weight
  - Supports many MPI implementations
  - Web page info
    https://www.nas.nasa.gov/hecc/support/kb/entry/525
  - Jul 20, 2016 Webinar slides
    https://www.nas.nasa.gov/hecc/support/past_webinars.html

- IOT
  - Brief introduction (software components and usage)
  - Demonstration with a few examples to highlight some capabilities

- Recommendation
  - Use mpiprof for a quick analysis
  - Followed by IOT for more in-depth analysis

What is IOT?

- A licensed toolkit from I/O Doctors, LLC.
- An I/O Instrumentation and Optimization tool
  - Provides details at various levels: per job, rank, file, I/O call
  - Reports time spent, number of calls, and number of bytes
  - **Can report when it happens, where in the file I/O occurs**
  - Can set and/or report stripe counts, stripe sizes
  - Can correlate I/O with system activities as a function of time, such as cputime, memory, network, Lnet, etc.
- Expanded (v4) to include MPI analysis and backtrace reporting
  - Works with multiple MPI implementations
  - Includes analysis of all MPI functions
  - Things reported for MPI functions are similar to those for I/O

* not available with NAS in-house mpiprof tool

Question? Use the Webex chat facility to ask the Host

Software Components of IOT

- **libiot.so**: a library for runtime interception of I/O functions
- **iot**: a Linux utility program
  - configures the runtime environment for instrumentation

    iot –m mpiflavor –f cfg.icf –j ${PBS_JOBID} mpiexec –np a.out

  - produces *iot.jobid.ilz*

    By default, instrumentation output streams from all processes are merged.

- **Pulse**: a Java based GUI for post-mortem analysis of the instrumentation streams generated by libiot.so
  - *pfe% module load jvm/jrel.8.0_121*

    pfe% pulse iot.jobid.ilz (or java –jar Pulse.jar iot.jobid.ilz)

  - On your desktop, download /nasa/IOT/latest/java/Pulse.jar and iot.jobid.ilz

    your-desktop% java –jar Pulse.jar iot.jobid.ilz (faster response than pfe)

Sample cfg.icf

```
#meminfo.interval={1000}
#cpustat.interval={1000}
#netstats.interval={1000}.devices={"*"}
#diskstats.devices={"sda:sdb:sdc"}.interval={1000}
#lnetstats.interval={1000}
#oscstats.interval={1000}.devices={""}
```

Can monitor system activities such as meminfo, cpustat, etc. at 1000 ms interval
means it is a comment, disabled

```
MPI.logInterval={((MPI_RANK%100==0)&&FUNC=="*")?-1:-2}
#MPI.backtrace={FUNC=="*"?5:0}
trc.logInterval={((MPI_RANK%100==0)&&FUNC=="*")?-1:-2}
#trc.backtrace={FUNC=="*"}
iol.name={PROGRAM+"."+PID}
PROGRAMS.include={"*"}
FILES.include={"*"}
LAYERS.use={trc,psx,lustre.stripeInfo={1}}
```

Flexible to choose what to track, such as MPI and/or I/O (trc), MPI_RANK, FUNCTIONs, level of details, PROGRAMS, Files, and Layers

Generic icf files under /nasa/IOT/latest/icf

- **trc_summary.icf** and **mpi_summary.icf** (logInterval=-1)

  Provide aggregated statistics on the total count, time spent, bytes transferred of **each function (and each file for trc_summary)** by **each MPI rank**

  Start with these when learning to use IOT

- **trc_interval.icf** and **mpi_interval.icf** (loginterval=1000)

  In addition to the information gathered with the xxx_summary.icf, this experiment provides additional details for each {I/O; MPI} function **the walltime when the calls occur**, counts, time spent and bytes transferred **per 1000 ms interval**

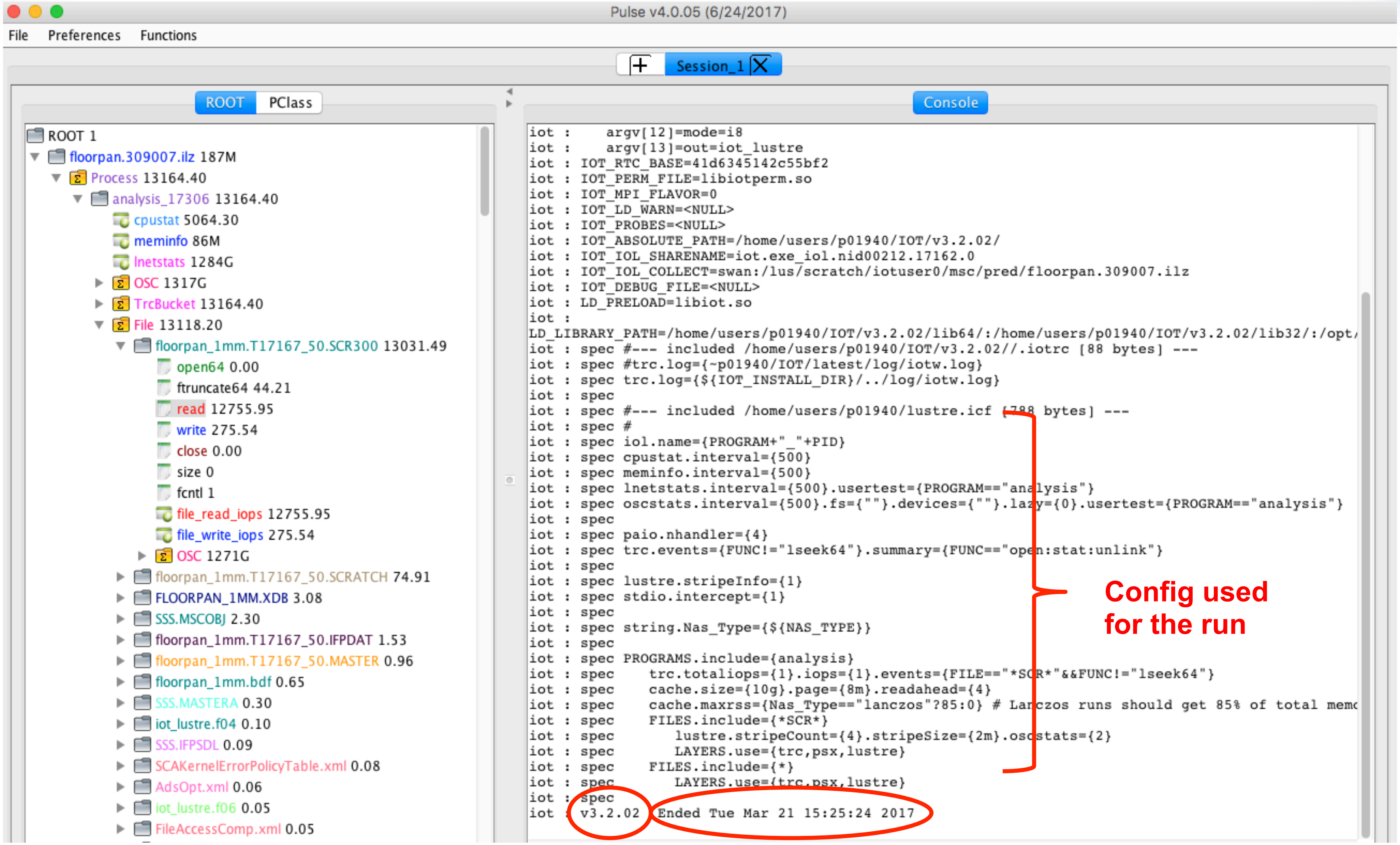
- **trc_events.icf** and **mpi_events.icf** (loginterval=0)

  In addition to the information gathered with the xxx_summary.icf, this experiment provides the greatest details for each {I/O;MPI} function the walltime **when each call occurs**, the time spent for the call and the number of bytes transferred

  Do not use until you have some experiences

Viewing Resulting Streams (ilz) via Pulse

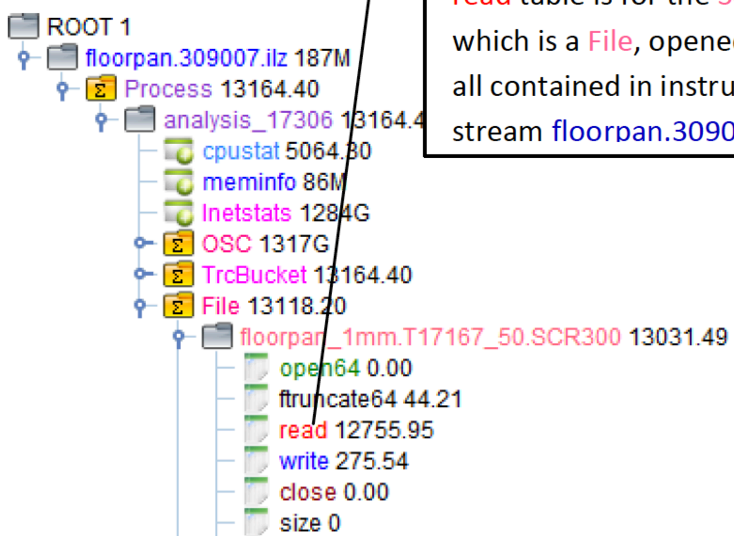
java -jar Pulse.jar floorpan.309007.ilz



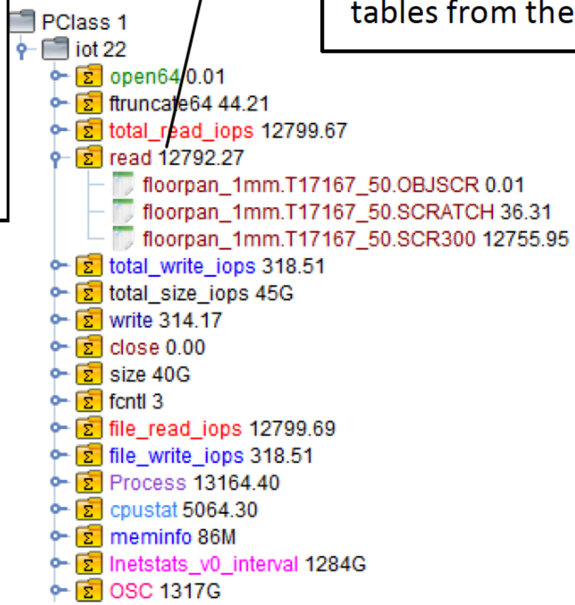
Question? Use the Webex chat facility to ask the Host

Pulse Provides Three Formats to Present Results
1. Hierarchical Tree Format

Entries in ROOT tree represent parent child relationship. This read table is for the SCR300 reads, which is a File, opened by analysis, all contained in instrumentation stream floorpan.309007.ilz



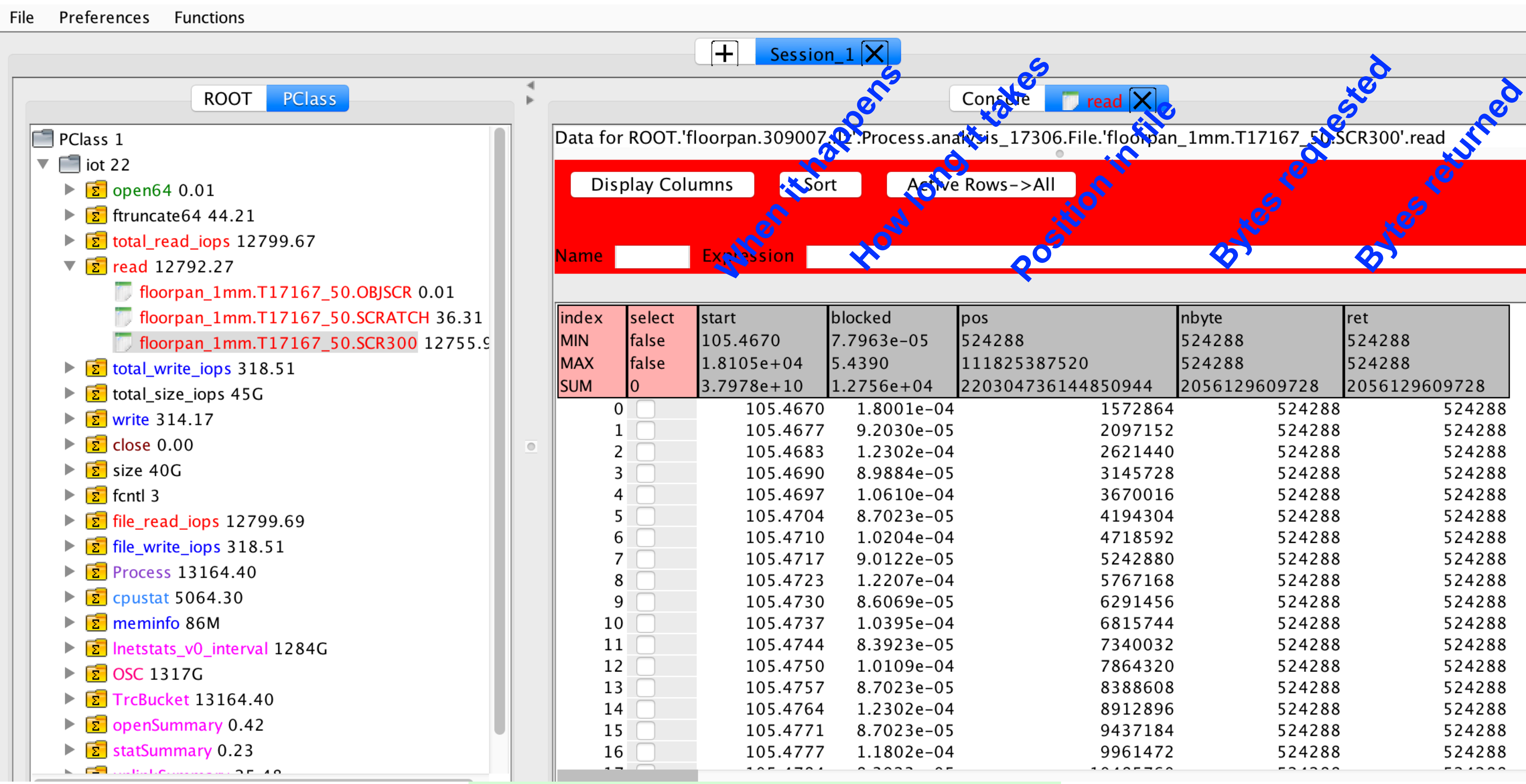
Entries in the PClass tree are arranged by like type. The read folder contains all read tables from the different files.



Question? Use the Webex chat facility to ask the Host

2. Table Format

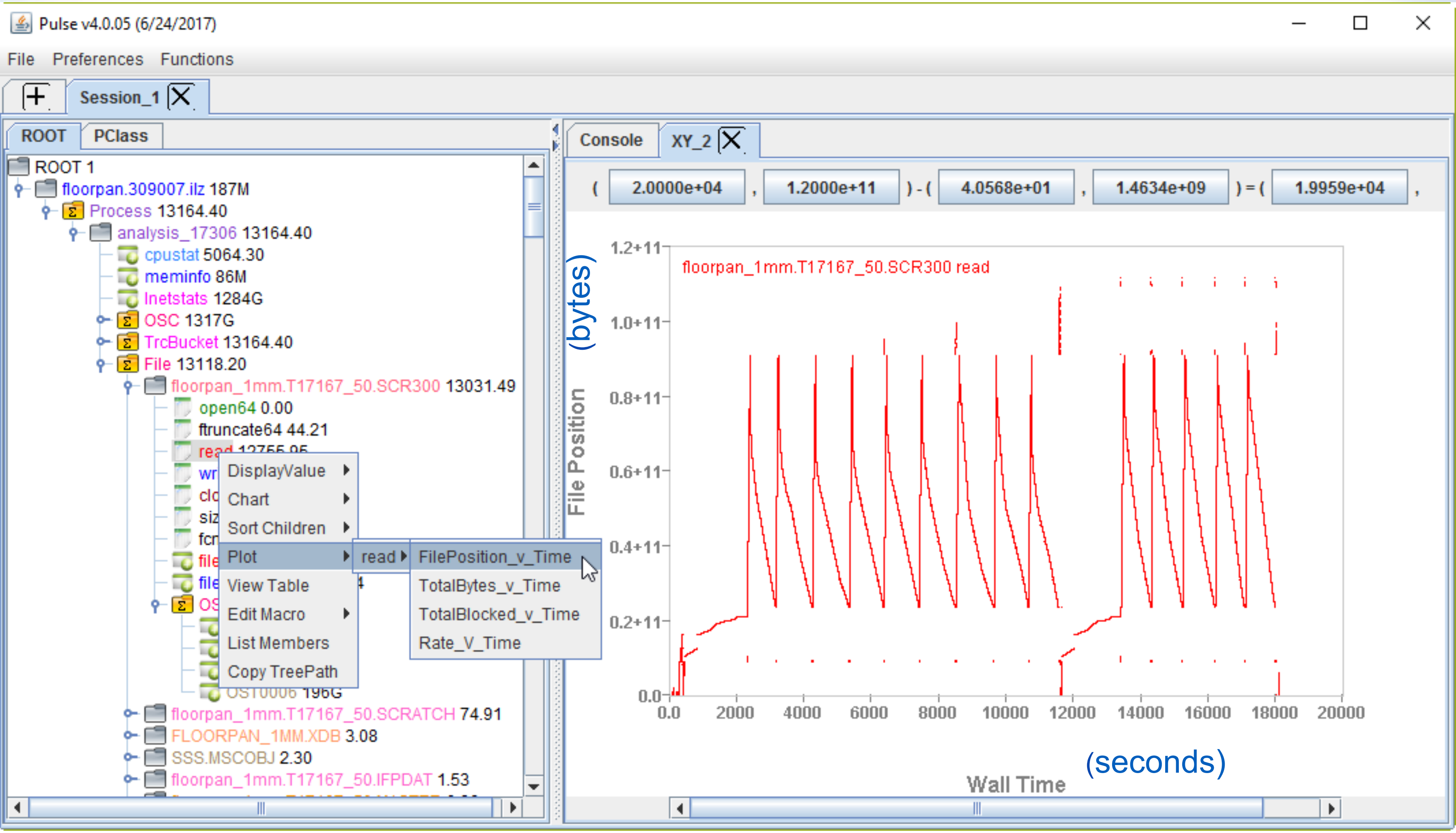
This is the start of the read table for the SCR300 file shown in the tree format. The column **blocked** is the amount of time spent in the individual read, **pos** is the file position at the start of the read, **nbyte** is the number of bytes requested by the read, and **ret** is the return value of the read function call.



index	select	start	blocked	pos	nbyte	ret
MIN	false	105.4670	7.7963e-05	524288	524288	524288
MAX	false	1.8105e+04	5.4390	111825387520	524288	524288
SUM	0	3.7978e+10	1.2756e+04	220304736144850944	2056129609728	2056129609728
0		105.4670	1.8001e-04	1572864	524288	524288
1		105.4677	9.2030e-05	2097152	524288	524288
2		105.4683	1.2302e-04	2621440	524288	524288
3		105.4690	8.9884e-05	3145728	524288	524288
4		105.4697	1.0610e-04	3670016	524288	524288
5		105.4704	8.7023e-05	4194304	524288	524288
6		105.4710	1.0204e-04	4718592	524288	524288
7		105.4717	9.0122e-05	5242880	524288	524288
8		105.4723	1.2207e-04	5767168	524288	524288
9		105.4730	8.6069e-05	6291456	524288	524288
10		105.4737	1.0395e-04	6815744	524288	524288
11		105.4744	8.3923e-05	7340032	524288	524288
12		105.4750	1.0109e-04	7864320	524288	524288
13		105.4757	8.7023e-05	8388608	524288	524288
14		105.4764	1.2302e-04	8912896	524288	524288
15		105.4771	8.7023e-05	9437184	524288	524288
16		105.4777	1.1802e-04	9961472	524288	524288

Question? Use the Webex chat facility to ask the Host

3. Graphical Format



Question? Use the Webex chat facility to ask the Host

Strengths/Benefits

- No root permission needed (anyone can run it after basic setup)
- No recompiling or relinking (-g required if you want backtrace)
- Minimal changes to run scripts (see slide "Components of IOT")
- **Works with script-based programs (such as R, matlab, Python)**
- Great flexibility in choosing what to instrument and monitor (make changes to cfg.icf in slide "Sample cfg.icf")
- Low overhead (if you know what you are doing, and avoid generating un-necessary amount of data)

Question? Use the Webex chat facility to ask the Host

When You May Want to Use IOT

- You want to get a better understanding of or validating how your program's MPI and/or I/O works

- You want to improve I/O performance and scaling of a code

- You were told by NAS staff that your program is causing the IB network or Lustre to be slow and you want to know why the code behaves this way

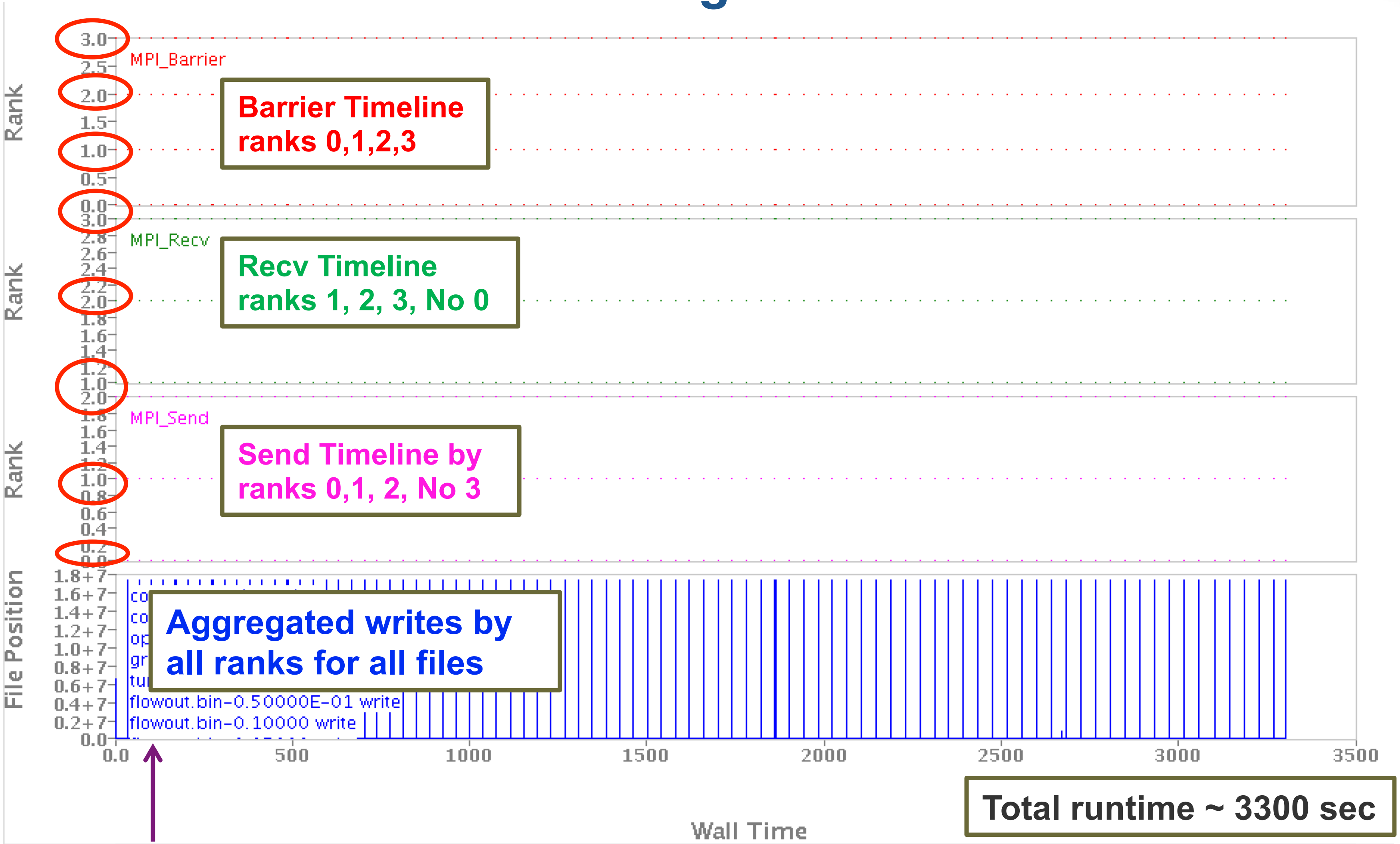
- You are porting a program from one system to another (hardware or software such as OS, MPI), and want to understand differences in MPI and/or I/O performance on the two systems

Four examples to demonstrate the use of IOT in each of these scenarios.

Question? Use the Webex chat facility to ask the Host

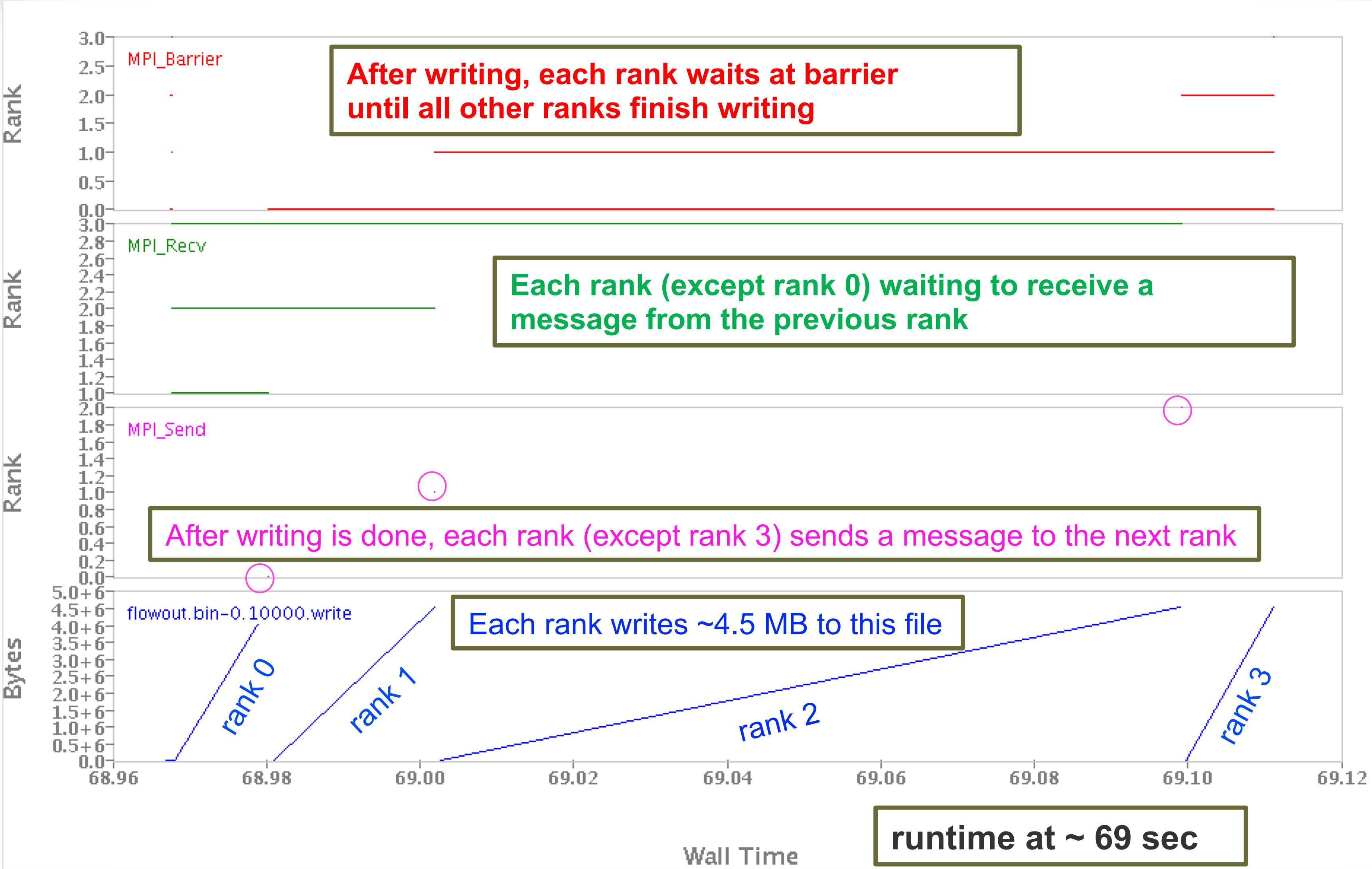
Example #1

Help to understand interplay between MPI and I/O for code #1 running with 4 MPI ranks



Tip: IOT GUI allows you zoom in to a region to see details, as in next slide

Example #1 (cont'd)
Zoom in to the time period when 1 file was being written



Proof of MPI_Send rank 0 ->1, 1 -> 2?

Example #1 (cont'd)

Table view of send and recv activities reveal additional details

Rank 1 waits for 0.0126 sec to receive 4 bytes message from rank 0 when rank 0 is done writing.

index	select	wallTime	count	wait	bytes	src	tag
MIN	false	2.6847	1	0.0100	4	0	10
MAX	false	3303.2176	1	0.0739	4	0	10
SUM	0	1.4044e+05	88	1.1473	352	0	880
0		2.6847	1	0.0739	4	0	10
1		2.7654	1	0.0123	4	0	10
2		35.6982	1	0.0124	4	0	10
3		68.9803	1	0.0126	4	0	10
4		102.7765	1	0.0122	4	0	10
5		136.5528	1	0.0120	4	0	10

index	select	wallTime	count	wait	bytes	dest	tag
MIN	false	2.6847	1	0.0	4	1	10
MAX	false	3303.2176	1	2.1458e-06	4	1	10
SUM	0	1.4044e+05	88	1.1730e-04	352	88	880
0		2.6847	1	1.9073e-06	4	1	10
1		2.7654	1	2.1458e-06	4	1	10
2		35.6982	1	9.5367e-07	4	1	10
3		68.9803	1	9.5367e-07	4	1	10
4		102.7765	1	9.5367e-07	4	1	10
5		136.5528	1	9.5367e-07	4	1	10

Rank 0 takes < 1 us to send 4 bytes message to rank 1 when rank 0 is done writing.

What we learned via IOT:
This code uses MPI send, recv and barrier to control the POSIX (non-MPI) writing of data to the same file by different MPI ranks.

The ability to correlate MPI and I/O activities can be useful for understanding some MPI codes.

IOT is much light-weight compared to Intel Trace Analyzer and Collector.

Example #2

Improve I/O Performance for Better Scaling for code #2

User found that I/O was taking much longer at larger core count which prevents scaling

Note: User's original set up used the 'then-default' stripe count of 1 for his runs
New stripe count default is 4

Example #2 (cont'd)

Per-file details provide great information

(5968 ranks, 1 stripe, tracking every file, every 280 ranks)

- IOL 3
  - Port 5000 0/0
  - athena2.collect.262099.iops.stripe_1.ilz
    - athena.48696.0 1148.26
      - cpustat 475.12
      - meminfo 435M
      - TrcBucket 1163.23
        - program_to_psx 1163.23
      - File 1148.26
    - athena.5942.5040 717.65
    - athena.16476.4760 681.07
    - athena.3248.280 666.75
    - athena.9866.3080 661.28
    - athena.64846.3640 652.09
    - athena.20945.840 637.47
    - athena.76175.4480 708.61
    - athena.97193.4200 663.02
    - athena.33546.3360 680.20
    - athena.53405.5600 752.26
    - athena.92814.2240 693.13
    - athena.5086.1680 635.68
    - athena.62068.1120 629.91
    - athena.91251.3920 675.50
    - athena.16877.5880 665.92
    - athena.51576.1400 677.58
    - athena.386.5320 655.20
    - athena.94851.2800 670.13

Data for TableTNO<TrcBucket>:program_to_psx[totalWait=1163.23]

Sort Display

Rank 0 Table

Name Expression

Tip: Columns can be reordered/removed for focused analysis

leafname	writeWait	openWait	readWait	closeWait
logT.txt	0.0	1.0459423065e-03	0.0	5.9604644775e-06
disk.out1.00000.athdf.xdmf	118.24443650	1.1972129345	0.02249908447	0.01133489609
741	1151.97	11.20	0.02	0.03
disk.out3.00000.rst	68.46667409	0.6926419735	0.0	2.4318695068e-04
disk.out3.00001.rst	93.42048359	0.7249491215	0.0	2.5892257690e-04
disk.out3.00002.rst	99.35387778	0.7809510231	0.0	0.01057505608
disk.out3.00003.rst	89.80697036	0.8424441814	0.0	3.6406517029e-04
disk.out3.00004.rst	94.07871604	0.7212369442	0.0	0.01133489609
disk.out3.00005.rst	87.79289865	0.8714570999	0.0	2.5296211243e-04
disk.out1.00000.athdf.xdmf	0.74988174	0.0929419994	0.0	2.5105476379e-04
disk.out1.00001.athdf.xdmf	0.54735327	0.0938870907	0.0	2.2697448730e-04
disk.out1.00002.athdf.xdmf	0.54726434	0.0257899761	0.0	1.9907951355e-04
disk.out1.00003.athdf.xdmf	0.78999877	0.0872049332	0.0	2.0003318787e-04
disk.out1.00004.athdf.xdmf	0.55018544	0.1223001480	0.0	2.4700164795e-04
disk.out1.00005.athdf.xdmf	0.54712081	0.0634109974	0.0	2.5606155396e-04
disk.out1.00000.athdf	89.19354653	1.1972129345	0.0	3.5047531128e-05
disk.out1.00001.athdf	112.94540977	0.8386909962	0.0	2.0980834961e-05
disk.out1.00002.athdf	82.14092469	0.6968750954	0.0	1.5974044800e-05
disk.out1.00003.athdf	118.24443650	0.8084239960	0.0	3.1948089600e-05
disk.out1.00004.athdf	112.81543517	0.7994279861	0.0	1.0013580322e-05
disk.out1.00005.athdf	99.98254800	0.7690601349	0.0	3.4093856812e-05
athinput.agndisk	0.0	0.0164110661	0.02249908447	5.9604644775e-06
aveopacity.txt	0.0	0.4421451092	0.0	3.3855438232e-05
logT.txt	0.0	0.2525751591	0.0	1.0013580322e-05

Sum = 532 sec

Sum ~ 4 sec

Sum = 615 sec

- **Main I/O files are: 6 rst, 6 athdf, and 6 xdmf files**
- **For each file, time spent in write > open > read and close**
- **Rank 0 write time: ~532 sec (6 rst), ~615 sec (6 athdf) >> ~4 sec (6 xdmf)**
- **rst and xdmf files written by rank 0 only; athdf files written by all ranks**
- **Non-0 ranks write time: 629 – 750 sec, mostly spent for writing the 6 athdf files**

Performance 1 stripe -> multiple stripes ?

Example #2 (cont'd)
I/O Improvement with larger stripe counts

5968 rank case; only rank 0 timing from IOT is shown here

File-level analysis not available with mpiprof

Time (sec)	Stripe 1	Stripe 16 (speedup)	Stripe 64 (speedup)	Stripe 128 (speedup)
Total I/O	1163.0	59.3 **(19.6)**	38.4 **(30.3)**	32.5 **(35.8)**
Total Write	1152.0	47.5 **(24.3)**	26.0 **(44.3)**	21.0 **(54.9)**
athdf Write	615.0	20.0 (30.8)	12.0 (51.3)	9.0 (68.3)
rst Write	532.0	25.0 (21.2)	8.0 (66.5)	3.0 (177.3)
xdmf Write	3.7	4.0 (0.9)	5.5 (0.7)	8.5 (0.4)

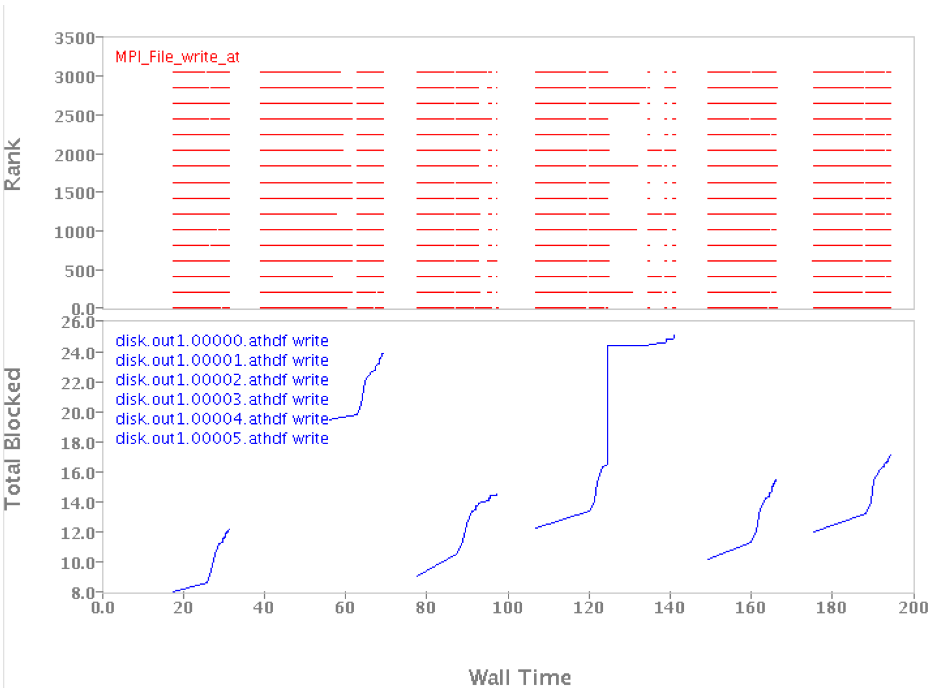
When the stripe count increases from 1 -> 16 -> 64 -> 128,

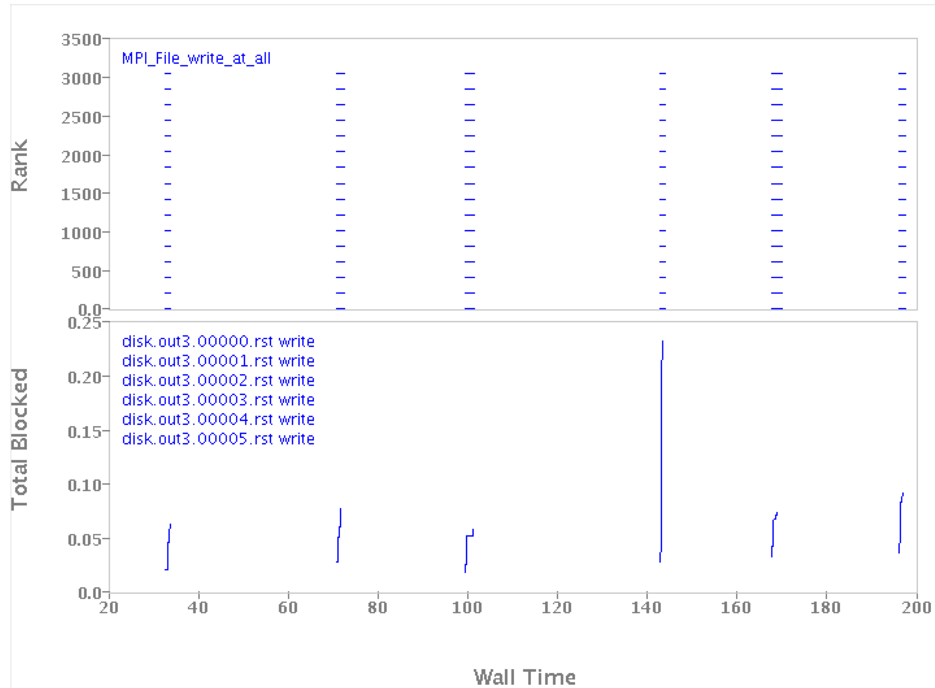
- get double digit speedup for the total I/O time and total write time
- speedup factor for writing the rst files is larger than that for the athdf files
- No speedup for writing the xdmf files

Tip: In the event that different files need very different stripe counts to get good performance, IOT provides a mechanism to set different stripe counts/sizes for different files in the IOT config file.

Example #2 (cont'd)
Digging deeper to understand difference in I/O for athdf and rst files

3264 rank case with 16 stripes; behavior for 5968 case is the same





Writing activities of the 6 athdf files correlate with the MPI_File_write_at function (non-collective);
Writing activities of the 6 rst files correlate with the MPI_File_write_at_all function (collective);
Seeing this direct correlation is possible with IOT v4 (which includes MPI monitoring);
Earlier versions do not have this capability.

Question? Use the Webex chat facility to ask the Host

Example #2 (cont'd)

What we learned via IOT:

- What write function is used for each file, amount of data, how many ranks write

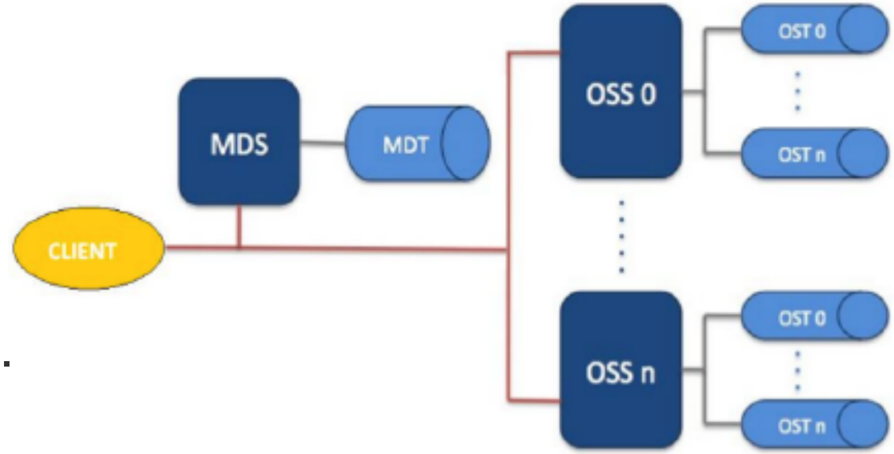
POSIX write from	Files	Bytes	Stripe 1 w-Ranks	Stripe 16 w-Ranks	Stripe 64 w-Ranks	Stripe 128 w-Ranks
MPI_File_write_at_all	6 rst	~323 G	0	16 Ranks	64 Ranks	128 Ranks
MPI_File_wirte_at	6 athdf	~14 G	all	all	all	all
Non-MPI write	6 xdmf	~ 60 K	0	0	0	0

- Why the rst files get the most speedup with increasing stripe counts:
  - large file sizes (~323 GB for rst vs ~14 GB for athdf)
  - increasing # of Lustre OSTs
  - increasing number of writing ranks (due to MPI collective I/O optimization)

- How MPI collective writes works (not covered in this webinar)

Example #3
Lustre is shared, bad I/O in a code affects others

NAS System Admins monitor the Lustre activities frequently and use system tools to identify code(s) that may be doing “bad things” to the Lustre filesystems. This requires catching the harmful activities in action.



Typical ”bad things” are
- Load imbalance in data access (such as r/w a very large file from/to a single OST)
- High frequency in checking file status (getcwd, stat)
- High frequency in opening/closing files (open, close)
- High frequency in repositioning read/write file offset (lseek)

If contacted by NAS staff, consider running job with IOT to know why.
The I/O activities of a PBS batch job are recorded in the ilz file and can be played back at any time.

Question? Use the Webex chat facility to ask the Host

Example #3 (cont'd)
Finding why high Lustre mds load occurs in code #3

index	select	leafname	openTime	openWait	readWait	writeWait	closeWait
MIN	false	logT.txt	5.4651	4.0102e-04	0.0	0.0	8.8215e-06
MAX	false	athinput.StellarEnveolp	599.4756	3.1601	0.0	0.0114	0.0069
SUM	0	1714	4.6476e+04	25.4568	0.0	0.5563	0.2547
0		athinput.StellarEnveolp	5.4651	0.0054	0.0	0.0	2.0909e-04
1		ghost.txt	6.5601	0.1954	0.0	0.0	2.2292e-04
2		aveopacity.txt	6.7566	0.0092	0.0	0.0	3.6001e-05
3		logT.txt	6.7661	0.0114	0.0	0.0	1.7881e-05
4		logRhoT.txt	6.7778	0.0356	0.0	0.0	8.8215e-06
5		Input.txt	6.5434	0.0166	0.0	0.0	2.6941e-05
6		Star.rad	7.0059	0.0065	0.0	7.1526e-06	1.9598e-04
7		Star.0000.rst	11.0386	0.0104	0.0	0.0062	0.0023
8		Star.hst	11.9686	0.1017	0.0	1.0000e-07	4.6301e-04
9		Star.0000.vtk	12.0718	0.1511	0.0	0.0015	0.0032
10		Star.0001.rst	25.3656	0.2335	0.0	0.0111	0.0025
11		Star.hst	28.9085	0.0064	0.0	0.0	1.5402e-04
12		Star.0001.vtk	28.9162	8.2803e-04	0.0	8.8247e-04	0.0013
13		Star.0002.rst	42.0544	0.2495	0.0	0.0106	0.0027
14		Star.hst	43.3174	0.7386	0.0	0.0	3.1495e-04
15		Star.0002.vtk	44.0577	0.5113	0.0	0.0015	0.0018
16		Star.0003.rst	58.0462	0.2418	0.0	0.0100	0.0034
17		Star.hst	58.4687	0.1009	0.0	0.0	2.5797e-04
18		Star.0003.vtk	58.5713	0.0072	0.0	0.0015	0.0019
19		Star.0004.rst	71.0723	0.2577	0.0	0.0100	0.0026
20		Star.hst	71.5513	0.0268	0.0	0.0	1.3995e-04
21		Star.0004.vtk	71.5789	0.0103	0.0	9.2172e-04	0.0012
22		Star.0005.rst	84.8015	0.2776	0.0	0.0100	0.0030
23		Star.hst	85.9297	0.0056	0.0	0.0	1.5903e-04
24		Star.0005.vtk	85.9361	8.4400e-04	0.0	8.5798e-04	0.0010

Test case has 2048 ranks, Only rank 0 data is shown here.

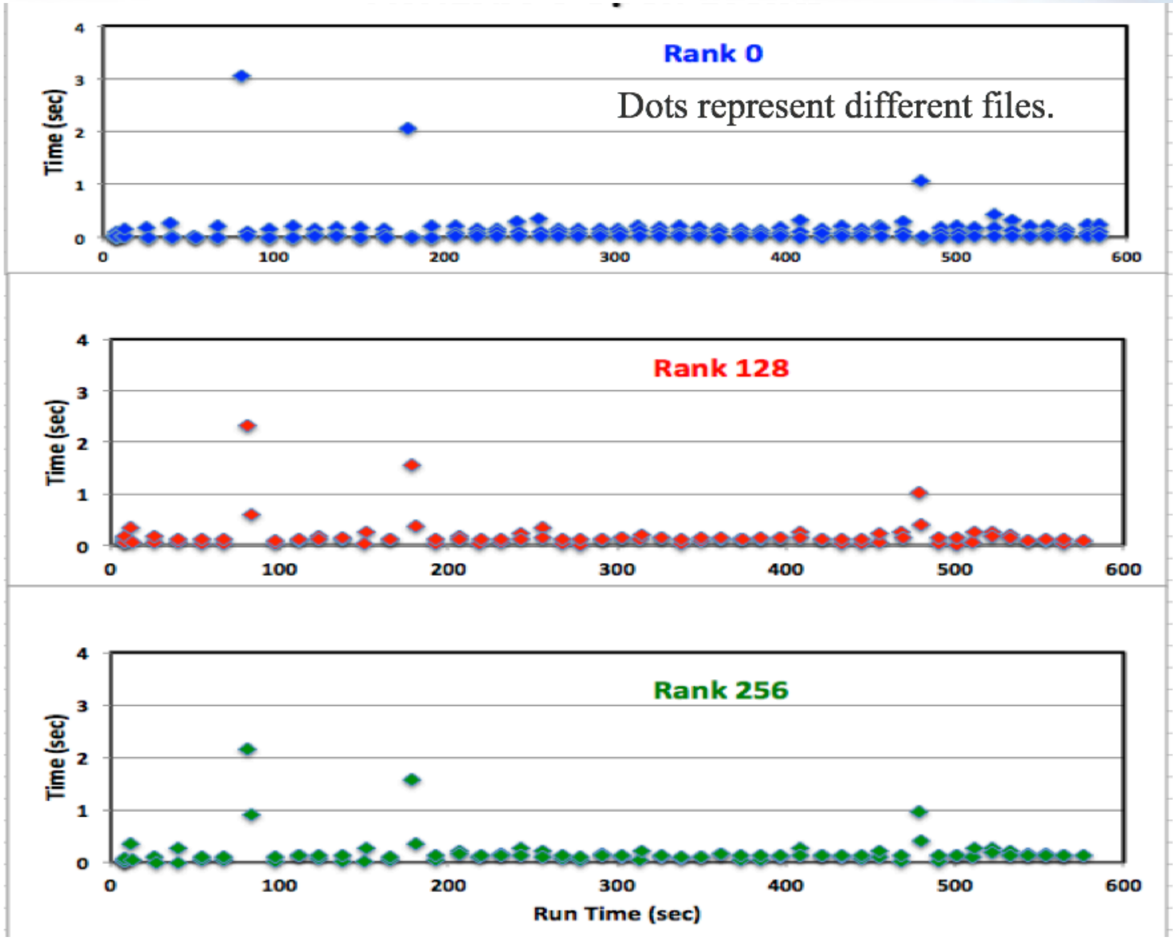
True for every rank, the per file I/O time spent in Open >> write > close

Example #3 (cont'd)

For this case,
each rank opens
its own set of 2-3 files
at a given time

Only 3 representative
Ranks are shown here

But the behavior is true
for all 2048 ranks



What we learned from IOT: Each one of the 2048 ranks in code #3 tries to open its own 2-3 files at the same time, with a total of >4000 open requests to MDS at a given time. When this happens at a high frequency, it may slow Lustre down.

Question? Use the Webex chat facility to ask the Host

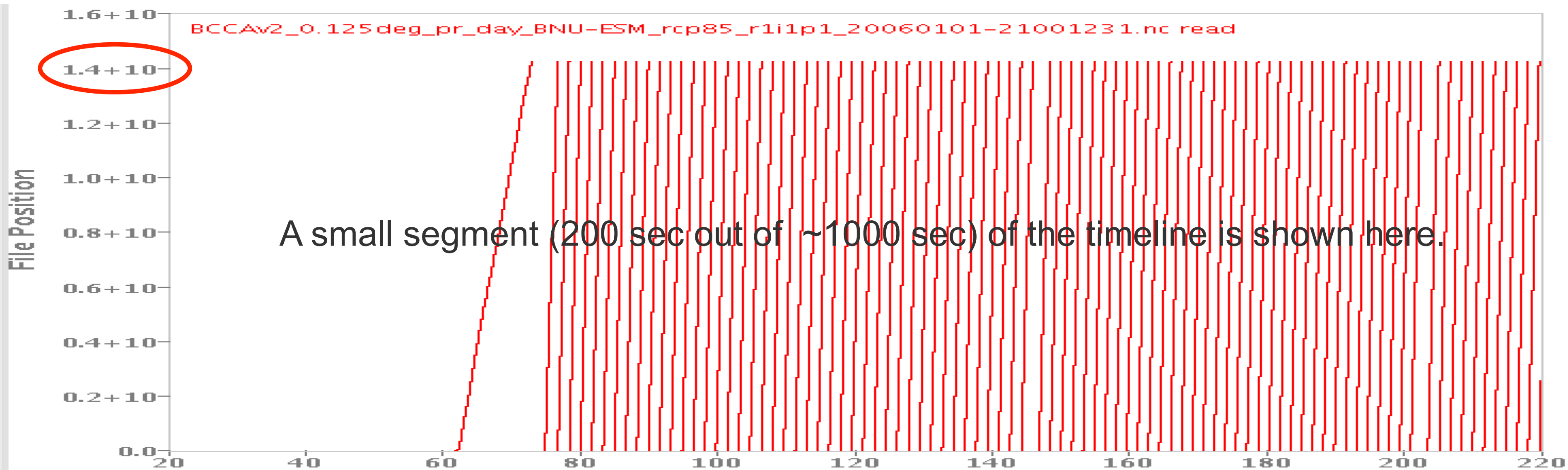
Example #4
Uncovering why I/O is much slower on Lustre than other filesystem(s) for code #4

- User's R application ran much slower on Lustre than laptop (Mar 2016)

- I/O analysis of R is straight-forward

  iot –f cfg.icf R CMD BATCH script.R

- IOT shows most of time is spent in reading a 14-GB netcdf file > 400 times



- No access to user's system; comparing behavior between Lustre and /tmp

Example #4 (cont'd)
I/O Difference in using Lustre and TMPFS (/tmp)

- ROOT 1
  - R_native_lustre.ilz 67M
    - Process 836.84
      - R.91975 836.84
        - cpustat 109.43
        - meminfo 1955M
        - Inetstats 14G
        - OSC 13G
        - TrcAggregate 836.84
          - program_to_psx 836.84
        - File 752.51
          - BCCAv2_0_125deg_pr_day_
            - open64 0.12
            - read 752.38
            - close 0.01
            - size 13G
          - OSC 301G

Data for ROOT.'R_native_lustre.ilz'.Process.'R.91975'.TrcAggregate.program_to_psx

Display Columns | Sort | Active Rows->All

Name | Expression

index MIN MAX SUM	select false false 0	fsType NFS Lustre 2549[illegible]8	origin fopen open64 213281	readCount 0 261338 37521424	readBytes 0 1020G 6191G	readBlocked 0.0 124.6523 808.3187
4559		Lustre	fopen	1	4	0.0816
4560		Lustre	open64	3394	13G	10.9312
4561		Lustre	fopen	1	4	0.0020
4562		Lustre	open64	145942	570G	69.2467
4563		Lustre	fopen	1	4	0.0020
4564		Lustre	open64	118790	464G	54.2850
4565		Lustre	fopen	1	4	0.0019
4566		Lustre	open64	261338	1020G	124.6523
4567		Lustre	fopen	1	4	0.0019
4568		Lustre	open64	78062	304G	36.3183
4569		Lustre	fopen	1	4	0.0020
4570		Lustre	open64	139154	543G	64.7186
4571		Lustre	fopen	1	4	0.0020
4572		Lustre	open64	162912	636G	76.5692
4573		Lustre	fopen	1	4	0.0019
4574		Lustre	open64	237580	928G	112.6093
4575		Lustre	fopen	1	4	0.0020
4576		Lustre	open64	200246	782G	92.2137
4577		Lustre	fopen	1	4	0.0020
4578		Lustre	open64	115396	450G	54.5975
4579		Lustre	fopen	1	4	0.0020
4580		Lustre	open64	122184	477G	56.1328

The file was opened/closed 11 times.
Each time it was opened, there were multiple reads from beginning to end.

- ROOT 1
  - R_native_tmp.2.ilz 570M
    - Process 73.36
      - R.14799 73.36
        - cpustat 133.98
        - meminfo 1955M
        - Inetstats 875M
        - OSC 0
        - TrcAggregate 73.36
          - program_to_psx 73.36
        - File 36.58
          - BCCAv2_0_125deg_pr_day_
            - open64 0.00
            - read 36.58
            - close 0.00
            - size 13G

Data for ROOT.'R_native_tmp.2.ilz'.Process.'R.14799'.TrcAggregate.program_to_psx

Display Columns | Sort | Active Rows->All

Name | Expression

index MIN MAX SUM	select false true 1	fsType NFS Lustre [illegible]	origin fopen open64 213281	readCount 0 2609K 49M	readBytes 0 20G 123G	readBlocked 0.0 6.0104 44.9072
4559		TMPFS	fopen	1	4	1.3113e-05
4560		TMPFS	open64	33K	271M	0.0787
4561		TMPFS	fopen	1	4	2.0027e-05
4562		TMPFS	open64	1457K	11G	3.3680
4563		TMPFS	fopen	1	4	1.7881e-05
4564		TMPFS	open64	1185K	9487M	2.7629
4565		TMPFS	fopen	1	4	2.1935e-05
4566		TMPFS	open64	2609K	20G	6.0104
4567		TMPFS	fopen	1	4	2.2888e-05
4568		TMPFS	open64	779K	6234M	1.7988
4569		TMPFS	fopen	1	4	2.1935e-05
4570		TMPFS	open64	1389K	10G	3.2231
4571		TMPFS	fopen	1	4	2.3127e-05
4572		TMPFS	open64	1626K	12G	3.7437
4573		TMPFS	fopen	1	4	2.3842e-05
4574		TMPFS	open64	2371K	18G	5.4424
4575		TMPFS	fopen	1	4	2.1935e-05
4576		TMPFS	open64	1999K	15G	4.6862
4577		TMPFS	fopen	1	4	2.2173e-05
4578		TMPFS	open64	1152K	9216M	2.6491
4579		TMPFS	fopen	1	4	2.1935e-05
4580		TMPFS	open64	1219K	9758M	2.8142

Less data read on TMPFS (123 GB) than Lustre (6191 GB)

Tip: one can load multiple ilz files in a single pulse session for easier comparison

Example #4 (cont'd)
I/O Difference in using Lustre and TMPFS (/tmp)

Lustre

index	select	start	blocked	pos	nbyte	ret
MIN	false	61.8420	4.4489e-04	0	4	4
MAX	false	843.3523	0.1241	13G	4M	4M
SUM	0	7.2777e+08	752.3758	10257T	6191G	6191G
0		61.8420	0.0816	0	4	4
1		61.9280	0.0033	0	4M	4M
2		61.9372	0.0295	4M	4M	4M
3		61.9668	0.0305	8M	4M	4M
4		61.9973	0.0307	12M	4M	4M
5		62.0281	0.0573	16M	4M	4M
6		62.0854	0.0309	20M	4M	4M
7		62.1163	0.0408	24M	4M	4M
8		62.1571	0.0202	28M	4M	4M
9		62.1773	0.0115	32M	4M	4M

TMPFS

index	select	start	blocked	pos	nbyte	ret
MIN	false	6.9384	9.5367e-07	0	4	4
MAX	false	161.9454	2.7204e-04	13G	8K	8K
SUM	0	1.3866e+09	36.5776	104897T	123G	123G
0		6.9384	1.3113e-05	0	4	4
1		6.9390	5.0068e-06	0	8K	8K
2		6.9417	5.9605e-06	400K	8K	8K
3		6.9417	3.0994e-06	800K	8K	8K
4		6.9417	1.9073e-06	1200K	8K	8K
5		6.9417	2.1458e-06	1608K	8K	8K
6		6.9417	1.9073e-06	2008K	8K	8K
7		6.9417	2.1458e-06	2408K	8K	8K
8		6.9417	2.8610e-06	2808K	8K	8K
9		6.9417	3.0994e-06	3208K	8K	8K

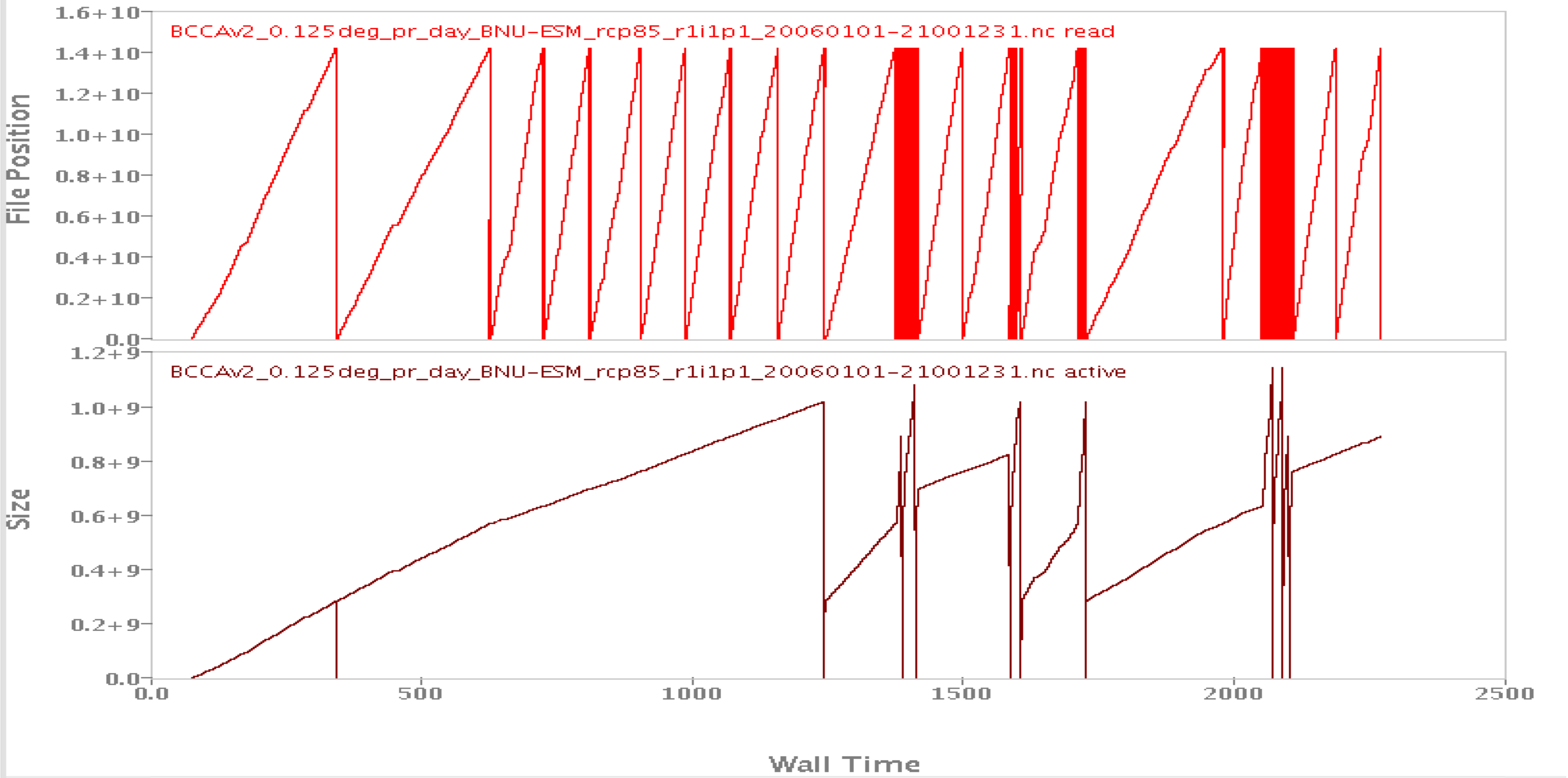
On Lustre, data were read in 4MB chunks from beginning to end
On TMPFS; data were read in 8KB chunks which are 400 KB apart
Somehow, the application is making each read size no less than the sector size of the filesystem used.
50 times more data were read when using Lustre
6191 GB/ 123 GB = 50
400 KB/8 K = 50

Showcase #4 (cont'd)
IOT provides proof that not all data are used

trc.**activeset**={8k}   keeping track of how many 8K chunks are actually touched by program.

…

LAYERS.use={trc,psx.sector={8k},lustre}



Question? Use the Webex chat facility to ask the Host

Showcase #4 (cont'd)

IOT can be used to assist with improving Lustre I/O without code changes

cache.page={4m}.size={20g}.readahead={4}.retain={1}.stdio={FILE="*.nc"}.zeropage={0}

paio.nhandler={4}

LAYERS.use={trc,cache,trc,paio,**psx.sector={8k}**,lustre.stripeInfo={3}}

- ROOT 1
  - R_cache_zeropage=0.ilz 555M
    - Process 61.40
      - R.99883 61.40
        - cpustat 151.98
        - meminfo 15G
        - lnetstats 922M
        - OSC 25K
        - TrcAggregate 61.40
          - program_to_psx 32.20
          - cache_to_paio 0.04
          - program_to_cache 29.16
            - readAggregate 27.68
            - statAggregate 0.00
            - openAggregate 0.24
        - Cache_v1 0
        - File 28.09
          - BCCAv2_0.125deg_pr_day_
            - open64 0.24
            - read 27.68
            - close 0.17
            - size 13G
            - BCCAv2_0.125deg_pr_

Data for ROOT.'R_cache_zeropage=0.ilz'.Process.'R.99883'.TrcAggregate.program_t

Display Columns | Sort | Active Rows->All

Name | Expression

index	select	fsType	origin	readCount	readBytes	readBlocked
MIN	false	Lustre	fopen	1	4	2.8610e-06
MAX	false	Lustre	open64	2671746	20G	7.4718
SUM	0	132	121	16203977	123G	27.6817
0		Lustre	fopen	1	4	0.0026
1		Lustre	open64	34698	271M	7.4718
2		Lustre	fopen	1	4	2.8610e-06
3		Lustre	open64	1492014	11G	1.8525
4		Lustre	fopen	1	4	3.8147e-06
5		Lustre	open64	1214430	9487M	1.4994
6		Lustre	fopen	1	4	4.0531e-06
7		Lustre	open64	2671746	20G	3.3476
8		Lustre	fopen	1	4	4.0531e-06
9		Lustre	open64	798054	6234M	0.9817
10		Lustre	fopen	1	4	3.0994e-06
11		Lustre	open64	1422618	10G	1.7721
12		Lustre	fopen	1	4	4.0531e-06
13		Lustre	open64	1665504	12G	2.0974
14		Lustre	fopen	1	4	3.0994e-06
15		Lustre	open64	2428860	18G	3.0574
16		Lustre	fopen	1	4	4.0531e-06
17		Lustre	open64	2047182	15G	2.5910
18		Lustre	fopen	1	4	3.0994e-06
19		Lustre	open64	1179732	9216M	1.4728
20		Lustre	fopen	1	4	3.8147e-06
21		Lustre	open64	1249128	9758M	1.5352

index	select	start	blocked	pos	nbyte	ret
MIN	false	7.2515	1.0000e-07	0	4	4
MAX	false	147.9844	0.0117	13G	8K	8K
SUM	0	1.3345e+09	27.6817	104897T	123G	123G
0		7.2515	0.0026	0	4	4
1		7.2561	3.0994e-06	0	8K	8K
2		7.2592	1.9073e-06	400K	8K	8K
3		7.2592	9.5367e-07	800K	8K	8K
4		7.2592	1.0000e-07	1200K	8K	8K
5		7.2592	9.5367e-07	1608K	8K	8K
6		7.2592	9.5367e-07	2008K	8K	8K
7		7.2592	9.5367e-07	2408K	8K	8K
8		7.2592	9.5367e-07	2808K	8K	8K
9		7.2592	9.5367e-07	3208K	8K	8K

Additional Info

- IOT quickstart guide for NAS users
  http://www.nas.nasa.gov/hecc/support/kb/entry/546
  Includes basic setup steps not covered in the slides

- Man pages
  pfe% /nasa/IOT/latest/bin64/iot –h (for iot options)
  pfe% /nasa/IOT/latest/bin64/iot –M (for layers options)

- IOT Documentation in pdf
  pfe% cd /nasa/IOT/Doc

- Request help with IOT by sending an email to support@nas.nasa.gov

Question? Use the Webex chat facility to ask the Host